LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-JC-152980

An Abstract Description Approach to the Discovery and Classification of Bioinformatics Web Sources

D. Rocco and T. Critchlow

May 1, 2003

The Fourth Georgia Tech and University of Georgia International Conference on Bioinformatics, Atlanta, Georgia, November 13-16, 2003

Updated October 14, 2003

An Abstract Description Approach to the Discovery and Classification of Bioinformatics Web Sources*

Daniel Rocco
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Terence Critchlow
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94551

Abstract

Motivation: The World Wide Web provides an incredible resource to genomics researchers in the form of dynamic data sources—e.g. BLAST sequence homology search interfaces. The growth rate of these sources outpaces the speed at which they can be manually classified, meaning that the available data is not being utilized to its full potential. Existing research has not addressed the problems of automatically locating, classifying, and integrating classes of bioinformatics data sources.

Results: This paper presents an overview of a system for finding classes of bioinformatics data sources and integrating them behind a unified interface. We examine an approach to classifying these sources automatically that relies on an abstract description format: the service class description. This format allows a domain expert to describe the important features of an entire class of services without tying that description to any particular Web source. We present the features of this description format in the context of BLAST sources to show how the service class description relates to Web sources that are being described. We then show how a service class description can be used to classify an arbitrary Web source to determine if that source is an instance of the described service. To validate the effectiveness of this approach, we have constructed a prototype that can correctly classify approximately two-thirds of the BLAST sources we tested. We then examine these results, consider the factors that affect correct automatic classification, and discuss future work.

Contact: rockdj@cc.gatech.edu; critchlow1@llnl.gov

*This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-ENG-48.

1 Introduction

The World Wide Web provides a mechanism for unprecedented information sharing among researchers. Today, scientists can easily post their research findings on the Web or compare their discoveries with previous work, often spurring innovation and further discovery. The value of accessing data from other institutions and the relative ease of disseminating this data has increased the opportunity for multi-institution collaborations, which produce dramatically larger data sets than were previously available and require advanced data management techniques for full utilization.

As a side effect of these types of collaborations, some tools become *de facto* standards in the communities as they are shared among a large number of institutions. For instance, consider the BLAST [1] family of applications, which allow biologists to find homologues of an input sequence in DNA and protein sequence libraries. BLAST is an example application that has been enhanced as a *Web source* that provides dynamic access to large data sets. Many genomics laboratories provide a Web-based BLAST interface [15, 9] to their sequence databases that allow scientists to easily identify homologues of an input sequence of interest. This capability enhances the genomics research environment by allowing scientists to compare new sequences to every known sequence and to have their work validated by other members of the community. The addition of new sequences at an increasingly frequent rate [16, 14] further increases the value of this capability.

Unfortunately, while the underlying program on many of these sites is the same, there is no common interface or data exchange mechanism for the established BLAST sources currently on the Web. To perform a BLAST search against multiple sources, a scientist must manually select the set of sites to query, enter their query into each site, and integrate the results. The problems with this approach are numerous: the scientist may not query the most relevant sites for their search, the search must be entered multiple times, the results of the search must be merged together by hand to obtain an integrated set of results, and if an interface changes or moves, the scientist must ascertain where the new interface is and how to query it appropriately.

Providing integrated access to BLAST Web sources is a challenging but important problem in genomics. The major challenges are to locate new Web sources, evaluate them to determine if they provide a BLAST interface, construct a wrapper for the source, and integrate the source into a mediator system that can provide a single point of access to all known sources conforming to the interface. Source autonomy complicates this problem: a cursory Web search yields hundreds of sources that provide a BLAST interface, many of which do not appear in bioinformatics directories [5]. Manually maintaining a wrapper library will not scale to accommodate the growth of genomics data sources on the Web, challenging us to produce an automated system that can find, classify, and wrap new sources without constant human intervention. This paper presents our approach for automatic classification of new Web sources into relevance categories that eliminates the human effort required to maintain a current repository of sources. The contributions of this paper are:

- A specification for describing classes of Web sources. These service class descriptions represent the salient features of the class concisely while providing enough information to distinguish instances of the class from other sources.

- A practical, heuristic approach to classifying arbitrary Web sources according to such a description.

This classification system is part of an ongoing research effort to build an automated integration system for Web sources. Our discussion in this paper focuses on BLAST interfaces to concretely demonstrate the approach, but these flexible techniques are generic and can be easily applied to other domains. In Section 2, we discuss existing research related to our work. We examine our service description format in Section 3 and the automatic classification system in Section 4. Section 5 describes our experimental evaluation using results obtained from applying our techniques to a set of BLAST sequence search services on the Web. As part of this discussion, we identify characteristics of sources that our prototype

cannot currently handle, which form the focus of ongoing work. We conclude with an examination of this work and future research opportunities.

2 Related Work

Our work is inspired by the ShopBot agent [6] whose purpose is to assist users in the task of online shopping. ShopBot uses the concept of a domain description that lists useful attributes about the services in question. The authors addressed the problems of learning unknown vendor sites and integrating a set of learned sources into a single interface. Our present work addresses the related problem of automatically classifying services from an arbitrary set of sites. The service class description format we describe provides greater descriptive power than ShopBot's domain descriptions and can specify complex data types and source control flow information.

Related to this work is the problem of heterogeneous data source integration. There are several research and commercial systems for querying heterogeneous data sources. Zadorozhny et al. [18] describe a wrapper and mediator system for limited-capability Web sources that includes query planning and rewriting capabilities. Information Manifold [12] targets the myriad Web interfaces to general purpose data, using a declarative source description for these sources combined with a set of query planning and optimization algorithms. The TSIMMIS [3] system provides mechanisms for describing and integrating diverse data sources while focusing on assisting humans with information processing and integration tasks.

Researchers have also examined heterogeneous data integration in the domain of biological data. DiscoveryLink [10] provides access to wrapped data sources and includes query planning and optimization capabilities. Eckman et al. [7] present a similar system with a comparison to many existing related efforts. BioKleisli [4] provides access to complex sources with structured data but does not include query optimization.

Our goal is to construct a system that can automatically discover and integrate bioinformatics Web sources. We seek to unify a class of sources such as BLAST behind a single interface that will maintain a current set of sources without manual intervention. This paper considers the classification aspect of this problem, which the above systems do not address. We expect that many of the techniques used by mediation systems will directly apply to the integration portions of our project.

Automatic discovery of Web sources apropos to a particular domain involves locating sources and determining their relevance to the domain; this paper addresses the second problem. Locating sources in the context of the Web typically involves a crawler that treats sites as nodes in a graph connected by hyperlink edges. Starting from a set of root pages, a crawler traverses the graph in some order specific to its goals and processes the sites it encounters. Part of this process involves extracting new hyperlinks to crawl from the encountered sites. While simple on the surface, Web crawling presents several research and implementation challenges, many of which have been addressed in the literature and commercially [2, 13, 11]. There is active research into topic driven or focused crawlers; Srinivasan et al. [17] present such a crawler for biomedical sources that includes a treatment of related systems.

3 Service Class Descriptions

Our approach to discovery and classification of Web sources groups sources into *service classes* that share common functionality but not necessarily a common interface. Service classes are specified by a *service class description*, which is an XML description format that defines the relevant aspects of a category of Web sources from an application's perspective. The service class description format addresses several design challenges that explicitly target the source discovery problem and provides a general description of the type of source that is considered interesting. It defines the data types used to build the source's interface as well as any intermediate types that may appear in a source. It establishes the interface used by source class members and outlines intervening control points. Finally, the description lists examples that are employed during source evaluation.

The description provides a mechanism for encapsulating the important components of a particular source that are common to all members of the class

and is the mechanism for hiding insignificant differences between individual sources. We expect that service class descriptions will be crafted in conjunction with domain scientists interested in utilizing automatic source discovery for their own application areas. This frees each user to determine the important characteristics of their domain and customize the search according to their individual requirements.

The description format allows a class to be described generically while providing enough information to differentiate between a set of arbitrary Web sources. A further requirement is that service classes be straightforward to create, although this can be fulfilled by the creation of a graphical description construction tool. The service class description format has been designed to meet these requirements while allowing descriptions to be tailored to the particular needs of the application domain.

```
<type name="DNASequence"
      type="string"
      pattern="[GCATgcat-]+" />

<type name="AlignmentSequenceFragment" >
    <element name="AlignmentName"
             type="string"
             pattern="[:alpha:]+:" />
    <element type="whitespace" />
    <element name="m"
             type="integer" />
    <element type="whitespace" />
    <element name="Sequence"
             type="DNASequence" />
    <element type="whitespace" />
    <element name="n"
             type="integer" />
</type>
```

Figure 1: *Some nucleotide BLAST type definitions.*

3.1 Types

The first component of a service class description specifies the data types that are used by members of the service class. Types in this context are analogous to those in programming languages. They are used to describe the input and output parameters of a service class and any data elements that may be required during the course of interacting with a source. The service class type system is modeled after the XML Schema [8] type system and includes constructs for building atomic and complex types. *Atomic types* are simple valued data elements such as strings and integers. The type system provides several built in atomic types that can be used to create user-defined types defined by restriction. The `DNASequence` type in Figure 1 is an example of an atomic type defined by restriction in the nucleotide BLAST service class description.

Atomic types can be composed into *complex types*, which are formed by composition of basic types into larger units. Figure 1 shows the specification of a nucleotide BLAST alignment sequence fragment, which is a string similar to:

```
Query:  280 TGGCAGGCGTCCT 292
```

The above string in a BLAST result would be recognized as an `AlignmentSequenceFragment` and annotated as such for later analysis.

Composition of elements into complex types can be in the form of a simple sequence of elements, such as the `AlignmentSequenceFragment` definition. List definitions are also allowed using the constraints `minOccurs` and `maxOccurs`, which define the expected cardinality of a particular sub-element within a type. The `choice` operator allows types to contain a set of possible sub-elements from which one will match.

3.2 Control Flow

Although all members of a service class provide similar functionality, the mechanics of different Web sources are virtually unconstrained. nucleotide BLAST sites provide interfaces ranging in complexity from a single input parameter for the sequence to sites having multiple input parameters spread across several pages and several stages of results. Further, when evaluating a given site, an automatic discovery agent will not know a priori if the site is a member of the given service class; if it is not, the agent may spend considerable effort wandering through a complicated workflow on an unrelated source.

To confront this problem, the service class description format provides syntax for enumerating the expected navigational paths used by members of the service class. This *control flow* graph consists of a

set of states connected by edges. Each state has an associated type; data from a Web source against the type associated with the control flow states to determine the flow of execution of a source from one state to another. Our nucleotide BLAST service class description, for example, has a single start state that defines the type of start page a class member must contain: in this case, any member of the nucleotide BLAST service class must have a start page that includes an HTML form with at least one text entry field.

3.3 Examples

The final component of the service class description is the examples. Examples contain input arguments and can be executed against an instance of the service class. While not strictly necessary for articulating a service class, the examples play an important role during analysis of a source. An example can be used to check if a site accepts input as required by the service class. The examples may also include negative or null queries that are expected to produce no results. Negative queries are useful for both validation purposes and for differentiating between data elements and template or style information in a source's responses.

Figure 2 shows an example used in a nucleotide BLAST description that illustrates the components of an example argument. The attribute `required` states whether an argument is a required input for all members of the service class. In this case, all members of the nucleotide BLAST service class are required to accept a DNA sequence as input. The argument lists the type of the input as well as a value that is used during classification. The optional `hints` section of the argument supplies clues to the site classifier that help select the most appropriate input parameters on a Web source to match an argument. This example also includes an argument for the program type, which is specified as an optional argument since some BLAST sources do not have a program selector input.

The argument hints specify the expected input parameter type for the argument and a list of likely form parameter names the argument might match. Multiple name hints are allowed, and each hint is treated as a regular expression to be matched against the form parameters. These hints are written by the service class description writer using their observation of typical members of the service class. For example, a DNA sequence is almost always entered into a text input parameter, usually with "sequence" in its name. The DNA Sequence argument in a nucleotide BLAST service class would therefore include a name hint of "sequence" and an input hint of "text."

```
<example>
  <arguments>
    <argument required="true">
      <name>sequence</name>
      <type>DNASequence</type>
      <hints>
        <hint>sequence</hint>
        <inputType>text</inputType>
      </hints>
      <value>TTGCCTCACATTGTCACTGCAAAT
             CGACACCTATTAATGGGTCTCACC
      </value>
    </argument>

    <argument required="false">
      <name>BlastProgram</name>
      <type>string</type>
      <hints>
        <hint>program</hint>
      </hints>
      <value>blastn</value>
    </argument>
  </arguments>

  <result type="SummaryPage" />
</example>
```

Figure 2: *A nucleotide BLAST example.*

4 Source Classification

Once a service class description has been defined, an automatic discovery agent can begin identifying Web sources of interest. This process encompasses two steps: locating sources and determining if they are instances of a service class. We have concentrated on the issue of identifying members of the service class. Specifically, given an arbitrary Web site and a service class description, we determine if the Web site is a Web source conforming to the interface defined by the description.

Analysis of the site starts with the control flow graph,which anchors the classification to the subset of input types specified as start points for this service class. Each node in the graph is treated as a state in an automaton: a match against a source at the current state allows the analysis to proceed to a connected state. The service classifier begins the analysis of a Web source by attempting to match the start page of the source against one of the start nodes in the control flow graph. If no matches are found, the source cannot match the service class and is discarded. If the start page matches, the classifier generates a series of queries using the examples provided in the service class description. For each response, the classifier then follows the outbound links and tests the responses of the source against the possible states in the control graph. This process continues until the site matches against one of the end states in the control flow graph or there are no more possible queries to try. The current prototype implementation of this source classifier is limited to simple control graphs consisting of a start state and end state only; enhancing the prototypes processing capabilities is a focus of ongoing work.

The output of the analysis and classification process states whether the input Web source matched the service class. If the site is a match, the classifier also lists the steps used at each control state to produce the result. This output includes information about the meaning of form parameters in the source along with the values used for each parameter.

Figure 3 shows part of this process visually. The control flow graph for a nucleotide BLAST service appears at (a) while (b) shows a representation of the control flow of a typical nucleotide BLAST source. In the graphs, start and end states are represented with circles and diamonds respectively, while interior points are shown with squares. The type associated with each state is listed inside it: t_a) nucleotide BLAST Input, t_b) formatting and query status, t_c) nucleotide BLAST Result Summary, and t_d) nucleotide BLAST Empty Result. If the Web source's start page (s) does not match the type of this control graph's start state (t_a), the classifier returns a negative result. If the start page matches, the classifier uses the examples to query the site, which returns its result (e). If this result is a nucleotide BLAST Result Summary or Empty Result, the type will match one of the end states (t_c) or (t_d). In this case, the classifier returns a positive result with the details needed to execute a query against the site. Note that although the state graph for the Web source is shown in Figure 3b, the service classifier will not know the control graph for the source beforehand. A significant part of the classifier's task is to infer this state graph using the attempted queries.

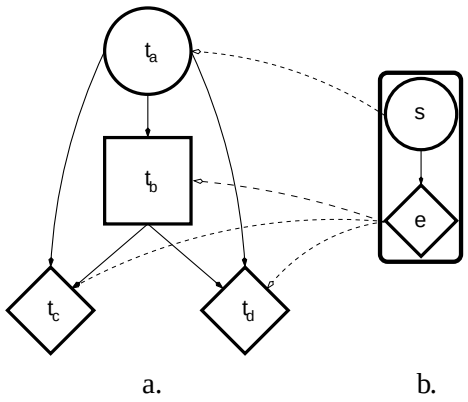

Figure 3: *Control Flow Graph for a nucleotide BLAST service (a) with potential Web source match (b).*

4.1 Query Generation

Generating queries with which to test a candidate service class member is a significant challenge when analyzing a Web source but is vital to verify whether the source is an instance of the service class. By probing a source using a variety of queries, different paths in the control flow graph can be explored to produce the subset of the control graph that models the source. Without generating queries with which to test a source, an analyzer could do little more than examine the source's form interfaces, which reveal very little about the nature of the underlying source. The query generator must take the examples from the service class description and produce a set of test queries that matches each argument in an example with a parameter in the sources's forms. Assume that a given source contains a set of forms F, where each form $f_i \in F$ has a set of parameters p_i. The number of possible queries Q for an example can be esti-

mated as

$$|Q| \approx \sum_{i=1}^{|F|} \binom{|p_i|}{a}$$

where a is the number of arguments in the example. Exhaustive search of the query space is clearly undesirable for sites with even a small number of form parameters: executing every query takes a significant amount of time and network bandwidth and is not likely to please the maintainers of the site being analyzed.

To combat these problems, we have developed a set of simple heuristics for choosing a small subset of Q to test that, despite their simplicity, work well for quickly determining class membership of real Web sources. These heuristics are based on four key observations:

1. Form parameters have type information
2. Parameters tend to be named purposefully
3. Parameters tend to have reasonable defaults
4. Output has common, recognizable components

Although HTML is not a data definition language, the parameters in a form reveal something about their expected values via their input tag's type attribute. Knowing that a particular parameter is a radio button limits the range of values that can be placed in it. The parameters also reveal something of their purpose by way of their name: on the overwhelming majority of sites used in our experiments, the name of the sequence input parameter contained the word "sequence." Many of the sites we examined expose a large subset of the options available to the BLAST program in there forms, but most of these parameters are set to default values that can be ignored without affecting a source's ability to produce results. The output format of the tested sources uses a consistent alignment format that can be recognized even when embellished with additional data.

Figure 4 displays the query generation process. A query enumerator combines the components from a service class description with the forms from a Web source. The output of the combination is a set of queries; the set contains a query for all pairings of the example's arguments with each parameter in all of the start page's forms. Each query is then assigned a score using a simple function that assigns points to a query for each parameter that matches the hints of its argument. Once a suitable ordering has been constructed, the queries will be executed in priority order until one leads to an end control state or there are no more queries to execute.

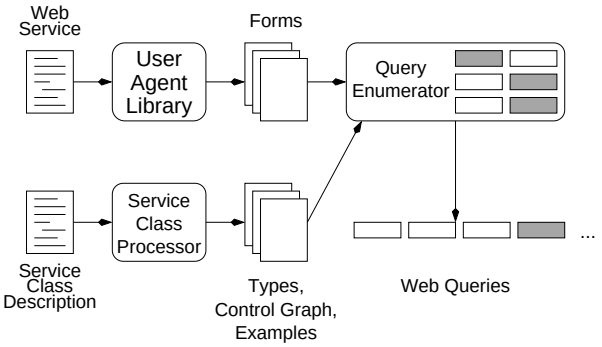

Figure 4: *Query generation process.*

5 Experimental Results

We have constructed a prototype of the source discovery system described here to test the validity of our approach. The prototype is implemented in Java and can examine a set of supplied URLs or crawl the Web looking for sources matching a description.

5.1 Methodology and Data

The data for our experiments consists of a list of 116 URLs that provide a BLAST interface that was gathered from the results of a Web search. The sites vary widely in complexity: some have forms with fewer than 5 input parameters, while others allow minute control over many of the options of the BLAST algorithm. Some of the sources, including the BLAST server at NCBI, include an intermediate step in the query submission process. A significant minority of the sources use JavaScript to validate user input or modify parameters based on other choices in the form. Despite the wide variety of styles found in BLAST sources, our prototype is able to recognize a large number of the sites using a service class description of approximately 150 lines.

Data Set	Successfully Identified	Failed Sites		Total	Percent Success
		Indirection	Processing		
Initial test set	18	5	4	27	66.7%
Experimental set	60	5	22	87	68.9%

Table 1: Sites classified using the nucleotide BLAST service class description.

We reserved a small subset of the nucleotide BLAST URLs for assisting the evaluation of the service class description language and our prototype implementation. The remainder of the sources were used for experimental testing after the prototype implementation was deemed ready. Sites that were manually determined to be non-functional or that returned results exclusively via email were excluded from our experiments and do not appear as part of the reported results.

5.2 Results

Table 1 shows the results of our experiments. The test set is the set of Web sources that were tested repeatedly as the prototype matured and helped shape its design. The remaining sources were categorized once. Sites listed as successes are those that can be correctly queried by the analyzer to produce an appropriate result, either a set of alignments or an empty BLAST result. An empty result indicates that the site was queried correctly but did not contain any homologues for the input sequence.

Failed sites are all false negatives that fall into two categories: indirection sources and processing failures. An indirection source is one that interposes some form of intermediate manual step between entering the query and receiving the result summary. For example, NCBI's BLAST server contains a formatting page after the query entry page that allows a user to tune the results of their query. Simpler indirection mechanisms include intermediate pages that contain hyperlinks to the results. We do not consider server-side or client-side redirection to fall into this category as these mechanisms are standardized and are handled automatically by Web user agents. Recognizing and moving past indirection pages presents several interesting challenges because of their free-form nature. Incorporating a general solution to complex, multi-step Web sources is part of our future work.

6 Conclusion

It is clear that the World Wide Web is an important tool for scientists and researchers. As the Web matures, we expect Web sources to continue proliferating while also adopting more robust data exchange standards like XML and RDF. We have explored the use of Web sources in the bioinformatics domain and seen that the increased number of sources promises greater research potential if the data management issues can be overcome. Our approach to this problem combines an abstract service class description with analysis techniques that map sources on the Web back to that description. We have shown how these concepts can be applied in an existing application scenario, Web-based BLAST genome sequence search. Finally, we have verified our claims experimentally by using a BLAST service class description to identify a group of Web sites.

Our initial results are very encouraging, as our categorization program consistently identified approximately two-thirds of the input URLs correctly. We attribute this success to the regularity of the returned data sets and the observed characteristics of Web sources. Of course, these preliminary results leave room for improvement. Many of the sources had complex interfaces that are not yet recognized by the prototype, which is limited to processing simple control graphs as noted previously. The remaining sources included sites that use JavaScript and a few with quirky interfaces. The prototype presently supports a subset of the full JavaScript specification and we are working to make our implementation more compliant with standard Web browser behavior.

We are continuing development of new heuristics for site processing and recognition. In particular, we

plan to expand the type handling system to identify such as indirection pages encountered during BLAST searches. The system will also be extended to support aggregation of data from hyperlinks—e.g. gene summaries commonly found in BLAST results. Longer term work will examine applying existing and novel information retrieval techniques to increase the number of recognized sources and further improve performance. For example, an advanced classification system could compare new sources to those it has already classified: if the new source matches a previously discovered source, the information from the existing match can be used to guide analysis of the new source.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215(3):403–410, October 1990.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.

[4] S. B. Davidson, G. C. Overton, V. Tannen, and L. Wong. BioKleisli: A digital library for biomedical researchers. *Int. J. on Digital Libraries*, 1(1):36–53, 1997.

[5] DBCAT, The Public Catalog of Databases. http://www.infobiogen.fr/services/dbcat/, 2002.

[6] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the world-wide web. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 39–48, Marina del Rey, CA, USA, 1997. ACM Press.

[7] B. Eckman, Z. Lacroix, and L. Raschid. Optimized seamless integration of biomolecular data. In *IEEE International Conference on Bioinformatics and Biomedical Egineering*, pages 23–32, 2001.

[8] D. C. Fallside. XML Schema Part 0: Primer. Technical report, World Wide Web Consortium, http://www.w3.org/TR/xmlschema-0/, 2001.

[9] W. Gish. BLAST. http://blast.wustl.edu/, 2002.

[10] L. Haas, P. Schwarz, P. Kodali, E. Kotlar, J. Rice, and W. Swope. Discoverylink: A system for integrating life sciences data. *IBM Systems Journal*, 40(2), 2001.

[11] A. Heydon and M. Najork. Mercator: A scalable, extensible web crawler. *World Wide Web*, 2(4):219–229, 1999.

[12] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proceedings of the Twenty-second International Conference on Very Large Databases*, pages 251–262, Bombay, India, 1996. VLDB Endowment, Saratoga, Calif.

[13] R. Miller and K. Bharat. SPHINX: A framework for creating personal, site-specific web crawlers. In *Proceedings of the Seventh International World Wide Web Conference*, 1998.

[14] National Center for Biotechnology Information. GenBank Statistics. http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html, 2003.

[15] National Library of Medicine/National Institutes of Health. National Center for Biotechnology Information. http://www.ncbi.nih.gov/, 2002.

[16] NIAS DNA Bank. Growth of daily updates of DNA Sequence Databases. http://www.dna.affrc.go.jp/htdocs/growth/D-daily.html, 2003.

[17] P. Srinivasan, J. Mitchell, O. Bodenreider, G. Pant, and F. Menczer. Web crawling agents for retrieving biomedical information. In *Proceedings of the International Workshop on Agents in Bioinformatics (NETTAB-02)*, 2002.

[18] V. Zadorozhny, L. Raschid, M.-E. Vidal, T. Urhan, and L. Bright. Efficient evaluation of queries in a mediator for websources. In *Proceedings of ACM/SIGMOD Annual Conference on Management of Data*, 2002.